Typing Linear Constraints for Moding CLP($\mathcal{R}$) Programs

Salvatore Ruggieri[1] and Fred Mesnard[2]

[1] Dipartimento di Informatica, Università di Pisa, Italy
ruggieri@di.unipi.it
[2] Iremia, Université de la Réunion, France
frederic.mesnard@univ-reunion.fr

Abstract. We present a type system for linear constraints over reals and its use in mode analysis of CLP programs. The type system is designed to reason about the properties of definiteness, lower and upper bounds of variables of a linear constraint. Two proof procedures are presented for checking validity of type assertions. The first one considers lower and upper bound types, and it relies on solving homogeneous linear programming problems. The second procedure, which deals with definiteness as well, relies on computing the Minkowski's form of a parameterized polyhedron. The two procedures are sound and complete. We extend the approach to deal with strict inequalities and disequalities. Type assertions are at the basis of moding constraint logic programs. We extend the notion of well-moding from pure logic programming to CLP($\mathcal{R}$).

Keywords: linear constraints, polyhedra, constraint logic programming, well-moding, definiteness.

1 Introduction

Modes in logic programming allow the user to specify the input-output behaviour of predicate arguments [1]. Modern constraint logic programming languages adopt moding both as program annotation and as a tool for compiler optimizations, program transformations and termination analysis. As an example, consider the `MORTGAGE` program over CLP($\mathcal{R}$).

```
(m1)  mortgage(P,T,R,B) ←          (m2)  mortgage(P,T,R,B) ←
          T = 0,                             T >= 1,
          B = P.                             NP = P + P * 0.05 - R,
                                             NT = T - 1,
                                             mortgage(NP,NT,R,B).
```

The query ← `mortgage(100, 5, 20, B)` is intended to calculate the balance of a mortgage of 100 units after giving back 20 units per year for a period of 5 years. The answer provides an exact value (i.e., a real number) for the required balance, namely `B = 17.12`. Using the moding terminology, we say that given *definite* values for principal, time and repayment, in every answer we

M. Alpuente and G. Vidal (Eds.): SAS 2008, LNCS 5079, pp. 128–143, 2008.

obtain a definite value for the balance. However, this is only one mode we can query the program above. The query ← `3 <= T, T <= 5, mortgage(100, T, 20, B)` is intended to calculate the balance at the end of the third, fourth and fifth year. Principal and repayment are now definite, whilst time is (upper and lower) *bounded*. Again, for every answer we will get a definite value for balance. Intuitively, this mode is more general than the previous one, since definiteness of time has been replaced by boundedness. Finally, consider the query ← `0 <= B, B <= 10, 15 <= R, R <= 20, mortgage(P, 5, R, B)`, which is intended to calculate the principal one could be granted such that by repaying from 15 to 20 units per year, after 5 years the balance yield is up to 10 units. The answer is now `P=0.78*B+4.33*R`, which is not definite, but, since `B` and `R` are bounded, it is (upper and lower) bounded. This mode is not comparable to the previous ones, since we now provide a definite value for time and a range for balance and repayment, and we wish to compute a range for the principal of the answer. These examples give only a few hints about the flexibility of the constraint logic programming scheme, even if compared to pure logic programming, where definiteness of variables corresponds to groundness, but upper and lower bounds have no direct equivalent.

In this paper, we concentrate on constraint languages with linear constraints over reals and rationals, as in CLP($\mathcal{R}$) [11], ECLiPSe, Sictus Prolog, SWI Prolog, and many others. We present a type system for linear constraints, where types model definiteness, upper and lower bounds of variables. Type assertions are introduced in order to derive types implied by a constraint and a set of typed variables. Validity of type assertions is thoroughly investigated by devising two proof procedures. The first one considers lower and upper bound types, and it relies on solving homogeneous linear programming problems. The second procedure, which deals with definiteness as well, relies on computing the Minkowski's form of a parameterized polyhedron. The two procedures are sound and complete. Moreover, the approach is extended to deal with constraints containing strict inequalities and disequalities. Type assertions are at the basis of moding CLP($\mathcal{R}$) programs. We extend the notion of well-moding from pure logic programming to CLP($\mathcal{R}$), showing useful properties in support of static analysis.

Preliminaries. We adhere to standard notation for linear algebra [16], linear programming [15] and (constraint) logic programming [1,10].

Linear Algebra. Small capital letters ($\mathbf{a}$, $\mathbf{b}$, ...) denote column vectors, while capital letters ($\mathbf{A}$, $\mathbf{B}$, ...) denote matrices. $\mathbf{0}$ and $\mathbf{1}$ are column vectors with all elements equal to 0 and 1 respectively. $\mathbf{a}_i$ denotes the i^{th} element in $\mathbf{a}$, and $row(\mathbf{A}, i)$ the row vector consisting of the i^{th} row of $\mathbf{A}$. $\mathbf{a}^T$ denotes the transposed vector of $\mathbf{a}$. $\mathbf{c}^T\mathbf{x}$ denotes the inner product of the transposed vector $\mathbf{c}^T$ and $\mathbf{x}$. $\Sigma\mathbf{v}$ is the sum of all the elements in $\mathbf{v}$. $\mathbf{Ax} \leq \mathbf{b}$ denotes a system of linear inequalities (or, a linear system) over the variables in $\mathbf{x}$. We assume that the dimensions of vectors and matrices in inner products and linear systems are of the appropriate size. The solution set of points that satisfy a formula/linear

system ψ over $\mathcal{R}^n$ is defined as $Sol(\psi) = \{\mathbf{x} \in \mathcal{R}^n \mid \psi(\mathbf{x})\}$. A polyhedron is the solution set of a linear system, namely $Sol(\mathbf{Ax} \leq \mathbf{b})$.

Linear Programming. A linear programming problem consists of determining $max\{\mathbf{c}^T\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b}\}$, if it exists. The problem is infeasible when $\{\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b}\} = \emptyset$. If feasible, but $\{\mathbf{c}^T\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b}\}$ has no upper bound, the problem is unbounded, and we write $max\{\mathbf{c}^T\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b}\} = \infty$. Otherwise, it is bounded. We write $max\{\mathbf{c}^T\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b}\} \in \mathcal{R}$ when the problem is feasible and bounded. We extend the notation to a closed set of points S by writing $max\{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} \in S\}$.

Constraint Logic Programming. The CLP Scheme defines a family of languages, CLP($\mathcal{C}$), that are parametric in the constraint domain $\mathcal{C}$. We are interested here in constraint domains over reals, such as CLP($\mathcal{R}$) [11]. All results apply to rationals as well. A primitive linear constraint is an expression $a_1 \cdot x_1 + \ldots a_n \cdot x_n \simeq a_0$, where $\simeq$ is in $\{\leq, =, \geq\}$, $a_1, \ldots, a_n$ are constants in $\mathcal{R}$ and $x_1, \ldots, x_n$ are variables. We will use the inner product form by rewriting it as $\mathbf{c}^T\mathbf{x} \simeq \alpha$. A linear constraint c is a sequence of primitive constraints, whose interpretation is their conjunction. A constraint logic program is a finite set of clauses of the form $A \leftarrow c, B_1, \ldots, B_n$, where A is an atom, c a linear constraint, and $B_1, \ldots, B_n$ $(n \geq 0)$ a sequence of atoms. We assume that atoms are in flat form, namely an atom is $p(x_1, \ldots, x_n)$ where p is a predicate of arity n and $x_1, \ldots, x_n$ are (not necessarily distinct) variables. A query $\leftarrow c, B_1, \ldots, B_n$ consists of a linear constraint and a sequence of atoms.

2 Bound Types for Linear Constraints

2.1 Syntax and Semantics

We introduce a static typing for variables in linear constraints. The set of types $\mathcal{BT}$ is defined first.

Definition 1 (types). *A* type *is an element of* $\mathcal{BT} = \{\star, \sqcup, \sqcap, \square, !\}$.

The intuitive meaning of a type is to label variables occurring in a constraint on the basis of the values that they can assume in the set of solutions of the constraint. ! is intended to type variables that show at most one single value in every solution, a property known as *definiteness*; $\square$ is intended to type variables that assume a range of values (hence, lower and upper bounds exist); $\sqcup$ (resp., $\sqcap$) is intended for variables that have a lower bound (resp., an upper bound); and finally, $\star$ is to be used when no upper or lower bound can be stated.

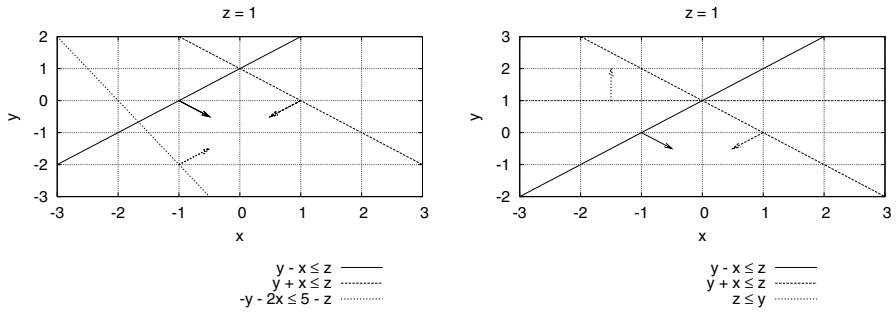
Let us introduce syntactic means to assert the type of variables.

Definition 2 (types assertions). *An* atomic type declaration *(atd, for short) is an expression $x : \tau$, where x is a variable and $\tau \in \mathcal{BT}$. We define $vars(x : \tau) = \{x\}$, and say that x is typed as τ. A* type declaration *is a sequence of atd's $d_1, \ldots, d_n$, with $n \geq 0$. We define $vars(d_1, \ldots, d_n) = \cup_{i=1..n} vars(d_i)$.*

A type assertion *is an expression $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$, where $\mathbf{d}_1, \mathbf{d}_2$ are type declarations and c is a linear constraint.*

Type declarations type variables. Such a typing is used in type assertions as an hypothesis (at the left of $\vdash$) or as a conclusion (at the right of $\rightarrow$). Intuitively, the type assertion $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ states that given the type declaration $\mathbf{d}_1$, the type declaration $\mathbf{d}_2$ holds under the linear constraint c.

Example 1. The type assertion $z :! \vdash y - x \leq z, y + x \leq z, -y - 2x \leq 5 - z \rightarrow y : \sqcap, x : \sqcup$ intuitively states that if z has a fixed value then the set of solutions of the involved constraint is such that y has an upper bound and x has a lower bound. The figure below (left) shows graphically the set of solutions for $z = 1$.



The type assertion $z :! \vdash y - x \leq z, y + x \leq z, z \leq y \rightarrow y :!, x :!$ states that if z has a fixed value then either the set of solutions of the involved constraint is empty or both x and y assume a unique value in it. The figure above (right) shows graphically the set of solutions for $z = 1$.

For a type declaration $\mathbf{d}$, we write $\mathbf{d}|_{\mathbf{x}}$ (resp., $\mathbf{d}|_{\tau}$) to denote the subsequence of $\mathbf{d}$ consisting only of atd's typing variables in $\mathbf{x}$ (resp., as τ). The intuition on the meaning of type assertions is formalized by the next definition.

Definition 3 (semantics). *We associate to an atd $d = x : \tau$ a formula $\phi(d)$ over fresh variables $\upsilon(d)$, called parameters, as follows:*

$$\begin{aligned} \phi(x :!) &= x = a & \upsilon(x :!) &= \{a\} \\ \phi(x : \square) &= a \leq x \wedge x \leq b & \upsilon(x : \square) &= \{a, b\} \\ \phi(x : \sqcup) &= a \leq x & \upsilon(x : \sqcup) &= \{a\} \\ \phi(x : \sqcap) &= x \leq b & \upsilon(x : \sqcap) &= \{b\} \\ \phi(x : \star) &= \texttt{true} & \upsilon(x : \star) &= \emptyset. \end{aligned}$$

ϕ and υ extend to type declarations as follows:

$$\phi(d_1, \ldots, d_n) = \wedge_{i=1..n} \phi(d_i) \qquad \upsilon(d_1, \ldots, d_n) = \cup_{i=1..n} \upsilon(d_i).$$

A type assertion $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ is valid if for $\mathbf{v} = vars(c) \cup vars(\mathbf{d}_1) \cup vars(\mathbf{d}_2)$, the following formula is true in the domain of reals:

$$\forall \upsilon(\mathbf{d}_1) \exists \upsilon(\mathbf{d}_2) \forall \mathbf{v}.(\phi(\mathbf{d}_1) \wedge c) \rightarrow \phi(\mathbf{d}_2). \tag{1}$$

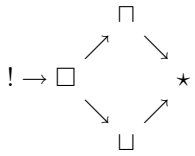
Example 2. For the type assertion $z :! \vdash y - x \leq z, y + x \leq z, z \leq y \rightarrow y :!, x :!$, the formula to be shown is:

$$\forall a\ \exists b, c\ \forall x, y, z.\ (z = a \wedge y - x \leq z \wedge y + x \leq z \wedge z \leq y) \rightarrow (y = b \wedge x = c).$$

The set of variables is fixed to $\mathbf{v} = vars(c) \cup vars(\mathbf{d}_1) \cup vars(\mathbf{d}_2)$ in order to take into account variables that appear in $\mathbf{d}_1$ or $\mathbf{d}_2$ but not in c, e.g. in the (valid) type assertion $x : \sqcap \vdash \texttt{true} \rightarrow x : \sqcap$.

A natural ordering over types is induced by the semantics above. For instance, it is readily checked that $\mathbf{d} \vdash c \rightarrow x :!$ implies $\mathbf{d} \vdash c \rightarrow x : \square$ for any $\mathbf{d}$, c and x. Similar implications lead to define an order $\geq_t$ over types.

Definition 4. *The $\geq_t$ partial order over $\mathcal{BT}$ is defined as the reflexive and transitive closure of the following relation $\rightarrow$:*



We write $\tau >_t \mu$ when $\tau \geq_t \mu$ and $\tau \neq \mu$. We define $lub(\emptyset) = \star$ and for $n > 0$:

$$lub(\{\tau_1, \ldots, \tau_n\}) = min\{\tau \mid \tau \geq_t \tau_i, i = 1..n\}.$$

Next, the $\geq_t$ relation is extended to type declarations.

Definition 5. *We write $\mathbf{d}_1 \geq_t \mathbf{d}_2$ if for every $x : \tau$ in $\mathbf{d}_2$ there exists $x : \mu$ in $\mathbf{d}_1$ such that $\mu \geq_t \tau$.*

Using the notation $\geq_t$, the intuition behind the ordering can be formalized by a monotonicity lemma. Also, transitivity is readily checked.

Lemma 1 (monotonicity). *Assume that $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ is valid. If $\mathbf{d}'_1 \geq_t \mathbf{d}_1$, $\mathbf{d}_2 \geq_t \mathbf{d}'_2$ and $\mathcal{R} \models c' \rightarrow c$ for a linear constraint c', then $\mathbf{d}'_1 \vdash c' \rightarrow \mathbf{d}'_2$ is valid.*

Normal forms for type declarations are introduced by assigning to each variable the least upper bound of its types. When the least upper bound is $\star$, the type assignment provides no actual information and then it can be discarded. Normal forms are unique modulo reordering of atd's.

Definition 6. *We define $nf(\mathbf{d})$ as any type declaration $\mathbf{d}'$ such that $x : \tau$ is in $\mathbf{d}'$ iff $\tau = lub(\{\mu \mid x : \mu$ is in $\mathbf{d}\ \})$ and $\tau \neq \star$.*

Example 3. Notice that $x : \square \geq_t x : \sqcap, x : \sqcup$ holds, while $x : \sqcap, x : \sqcup \geq_t x : \square$ does not hold. Actually, $\geq_t$ does not capture semantic implication. We have to move to normal forms to conclude that $nf(x : \sqcap, x : \sqcup) = x : \square \geq_t x : \square$.

Normal forms precisely characterize validity when it only depends on type declarations, i.e. for the constraint `true`.

Lemma 2. $\mathbf{d}_1 \vdash \texttt{true} \rightarrow \mathbf{d}_2$ *is valid iff* $nf(\mathbf{d}_1) \geq_t nf(\mathbf{d}_2)$.

2.2 Checking Type Assertions: First Intuitions

In principle, formulas as in (1) can be checked by real quantifier elimination methods [6], which trace back to Tarski's decision procedure for first order formula over real polynomials. However, while quantifier elimination represents a direct solution to the checking problem and it allows for generalizing to the non-linear case, we observe that formulas in (1) represent a quite restricted class. We will be looking for a specialized and efficient approach to check them. In addition, we are interested in the problem of inferring the largest (w.r.t. the $\geq_t$ order) $\mathbf{d}'$ such that $\mathbf{d} \vdash c \rightarrow \mathbf{d}'$ is valid, given $\mathbf{d}$ and c. Our approach switches from the *logical* view of constraints-as-formulas to a *geometric* view of constraints-as-polyhedra. Consider a linear constraint c and a type declaration $\mathbf{d}$. We observe that c can be equivalently represented as a linear system of inequalities $\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c$ where $\mathbf{v} = vars(c) \cup vars(\mathbf{d})$. The set of solutions of c coincides then with the polyhedron represented by $\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c$, which we call the *geometric* representation of c. Analogously, the linear constraint $\phi(\mathbf{d})$ can be represented as $\mathbf{A_d v} \leq \mathbf{B_d a_d}$, where $\mathbf{a_d}$ is the symbolic vector of parameters in $\upsilon(\mathbf{d})$. The resulting system $\phi(\mathbf{d}) \wedge c$ is a parameterized system of linear inequalities $\mathcal{P}$, where variables in $\upsilon(\mathbf{d})$ play the role of parameters:

$$\begin{pmatrix} \mathbf{A}_c \\ \mathbf{A_d} \end{pmatrix} \mathbf{v} \leq \begin{pmatrix} \mathbf{b}_c \\ \mathbf{0} \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ \mathbf{B_d} \end{pmatrix} \mathbf{a_d} \tag{2}$$

The notion of parameterized polyhedra models the solutions of parameterized linear systems.

Definition 7 (Parameterized polyhedron). *A* parameterized polyhedron *is a collection of polyhedra defined by fixing the value for parameters in a parameterized system of linear inequalities:* $Sol(\mathbf{Ax} \leq \mathbf{b}+\mathbf{Ba}, \mathbf{u}) = \{\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b}+\mathbf{Bu}\}$.

$Sol()$ is now a binary function. In addition to a system of parameterized linear inequalities, an assignment to parameters is required.

Example 4. Let $\mathbf{d}$ be z :! and c be $y - x \leq z, y + x \leq z, -y - 2x \leq 5 - z$. We have that $\phi(\mathbf{d})$ is $z = a$, and then the parameterized system for $\phi(\mathbf{d}) \wedge c$ is:

$$\begin{pmatrix} \text{-1} & \text{1} & \text{-1} \\ \text{1} & \text{1} & \text{-1} \\ \text{-2} & \text{-1} & \text{1} \\ \text{0} & \text{0} & \text{1} \\ \text{0} & \text{0} & \text{-1} \end{pmatrix} \begin{pmatrix} \text{x} \\ \text{y} \\ \text{z} \end{pmatrix} \leq \begin{pmatrix} \text{0} \\ \text{0} \\ \text{5} \\ \text{0} \\ \text{0} \end{pmatrix} + \begin{pmatrix} \text{0} \\ \text{0} \\ \text{0} \\ \text{1} \\ \text{-1} \end{pmatrix} a$$

Under this interpretation, validity of $\mathbf{d} \vdash c \rightarrow x : \tau$ has an intuitive geometric interpretation. Assume that $x = \mathbf{v}_i$. $\mathbf{d} \vdash c \rightarrow x : \tau$ is valid iff for every $\mathbf{u} \in \mathcal{R}^{|\upsilon(\mathbf{d})|}$, the set of solution points $\mathcal{S}_\mathbf{u} = Sol(\mathcal{P}, \mathbf{u})$ either is empty or:

- if $\tau = !$ then $max\{\mathbf{v}_i \mid \mathbf{v} \in \mathcal{S}_\mathbf{u}\} = min\{\mathbf{v}_i \mid \mathbf{v} \in \mathcal{S}_\mathbf{u}\} \in \mathcal{R}$, namely x assumes a single value;
- if $\tau = \square$ then $max\{\mathbf{v}_i \mid \mathbf{v} \in \mathcal{S}_\mathbf{u}\} \in \mathcal{R}$ and $min\{\mathbf{v}_i \mid \mathbf{v} \in \mathcal{S}_\mathbf{u}\} \in \mathcal{R}$ namely both an upper and a lower bound exist for x;
- if $\tau = \sqcup$ then $min\{\mathbf{v}_i \mid \mathbf{v} \in \mathcal{S}_\mathbf{u}\} \in \mathcal{R}$, namely a lower bound exists for x;
- if $\tau = \sqcap$ then $max\{\mathbf{v}_i \mid \mathbf{v} \in \mathcal{S}_\mathbf{u}\} \in \mathcal{R}$, namely an upper bound exists for x;
- if $\tau = \star$ then we have nothing to show.

Unfortunately, this procedure is not effective, since there are infinitely many $\mathcal{S}_\mathbf{u}$ to be checked. In the next two subsections, we will develop approaches for turning the intuitions above into effective and efficient procedures.

2.3 Checking Type Assertions: An LP Aproach

In this section we develop an inference algorithm which does not explicitly take into account parameters. We will be able to reason on type assertions over $\mathcal{BT} \setminus \{!\}$. First of all, let us consider the case of unsatisfiable constraints.

Lemma 3. *Consider the parameterized polyhedron $\mathcal{P}$ in (2). There exists a parameter instance* $\mathbf{u}$ *such that $Sol(\mathcal{P}, \mathbf{u}) \neq \emptyset$ iff $Sol(\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c) \neq \emptyset$.*

As a consequence, if $Sol(\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c) = \emptyset$ (i.e., c is an unsatisfiable constraint) then there is no chance to obtain a non-empty polyhedron by some instantiation of the parameters in $\phi(\mathbf{d})$. In this case, we can infer assertions of the form $\mathbf{d} \vdash c \rightarrow x :!$, for every variable x. From now on, we will concentrate then on satisfiable constraints. As it will be recalled later on, a non-empty polyhedron $Sol(\mathbf{Ax} \leq \mathbf{b})$ can be decomposed into the vectorial sum of its characteristic cone $Sol(\mathbf{Ax} \leq \mathbf{0})$ with a polytope, a polyhedra bounded along every dimension. Therefore, the existence of an upper/lower bound for a linear function over a polyhedron depends only on its characteristic cone. It is immediate to observe that for every parameter instance $\mathbf{u}$, the polyhedra $Sol(\mathcal{P}, \mathbf{u})$ share the same characteristic cone. As a consequence, proving the existence of an upper bound is independent from the parameter instance, and it relies only on the homogeneous version of $\mathcal{P}$, which is not anymore parameterized.

Lemma 4. *Consider the parameterized polyhedron $\mathcal{P}$ in (2). Let $\mathcal{H}$ be its homogeneous version: $\mathbf{A}_c\mathbf{v} \leq \mathbf{0}$, $\mathbf{A}_\mathbf{d}\mathbf{v} \leq \mathbf{0}$, and assume that $Sol(\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c) \neq \emptyset$.*

We have that $max\{\mathbf{c}^T\mathbf{v} \mid \mathbf{v} \in Sol(\mathcal{H})\} = 0$ iff for every parameter instance $\mathbf{u}$, $Sol(\mathcal{P}, \mathbf{u}) = \emptyset$ or $max\{\mathbf{c}^T\mathbf{v} \mid \mathbf{v} \in Sol(\mathcal{P}, \mathbf{u})\} \in \mathcal{R}$.

When $\mathbf{c}$ is always 0 except for the i^{th} position where it is 1, we have $\mathbf{c}^T\mathbf{v} = \mathbf{v}_i$. Lemma 4 solves then the problem of deciding whether $\mathbf{d} \vdash c \rightarrow \mathbf{v}_i : \sqcap$, without having to take into account parameters. By reasoning similarly for types $\sqcup$ and $\square$, we can state an effective procedure, called LPCheck and summarized in Fig. 1, which outputs type declarations in normal form without any trivial atd.

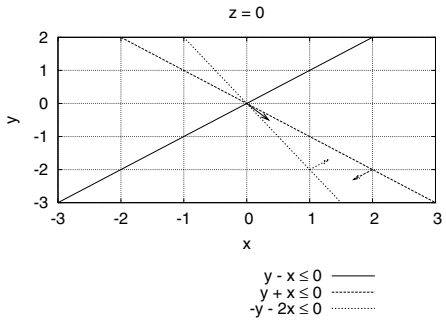
Example 5. The homogeneous version of the parameterized linear system in Example 4 and its graphical representation are the following:

Input: a type assertion $\mathbf{d}_1$, a linear constraint c and a sequence of variables $\mathbf{x}$.

Step 0 Define $\mathbf{v} = vars(c)$, $\mathbf{n} = nf(\mathbf{d}_1)$, $\mathbf{d} = \mathbf{n}|_{\mathbf{v}}$.
Step 1 Let $\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c$ be the geometric representation of c, and $\mathbf{A_d v} \leq \mathbf{B_d a_d}$ the geometric representation of $\phi(\mathbf{d})$.
Step 2 If $Sol(\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c) = \emptyset$ Then for every x in $\mathbf{x}$, output "x :!"
Else
Step 3 for every x in $\mathbf{x} \setminus \mathbf{v}$ and $x : \tau$ in $\mathbf{n}$, output "$x : \tau$";
Step 4 for every x in $\mathbf{x} \cap \mathbf{v}$:
(a) Let $M = max\{x \mid \mathbf{A}_c\mathbf{v} \leq \mathbf{0}, \mathbf{A_d v} \leq \mathbf{0}\}$ and $m = max\{-x \mid \mathbf{A}_c\mathbf{v} \leq \mathbf{0}, \mathbf{A_d v} \leq \mathbf{0}\}$.
(b) Output "$x : \square$" if $M = 0$ and $m = 0$;
(c) Output "$x : \sqcup$" if $M = \infty$ and $m = 0$;
(d) Output "$x : \sqcap$" if $M = 0$ and $m = \infty$.

Fig. 1. LPCheck procedure

$$\begin{pmatrix} -1 & 1 & -1 \\ 1 & 1 & -1 \\ -2 & -1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \leq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$



It is readily checked that x has a lower bound and y has an upper bound.

Soundness and a relative form of completeness of procedure LPCheck follow.

Theorem 1 (LPCheck - soundness and completeness). *Let* $\mathbf{d}_1$ *be a type declaration and* c *a linear constraint. If the sequence* $\mathbf{d}_2$ *is provided as output by* LPCheck*, the type assertion* $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ *is valid. Conversely, assume that* $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ *is valid, and that* c *is unsatisfiable or no variable in* $vars(c)$ *is typed as* ! *in* $\mathbf{d}_2$*. Then there exists a sequence* $\mathbf{d}$ *provided as output by* LPCheck *such that* $\mathbf{d} \geq_t nf(\mathbf{d}_2)$.

The LPCheck procedure is not tied to any underlying linear programming solver. By adopting a polynomial time algorithm [15,16], we can conclude that LPCheck has a polynomial time complexity. Due to the approach that we will follow later on for dealing with parameters, we present here an instantiation of LPCheck which consists of directly computing the generating matrix and the vertex matrix of polyhedra. This is an alternative representation of polyhedra, known as the explicit representation or the Minkowski's form [16, Section 8.9].

Theorem 2 (Minkowski's decomposition theorem for polyhedra). *There exists an effective procedure that given* $\mathbf{Ax} \leq \mathbf{b}$ *decides whether or not the*

polyhedron $Sol(\mathbf{Ax} \leq \mathbf{b})$ *is empty and, if not, it yields a generating matrix* $\mathbf{R}$ *and a vertex matrix* $\mathbf{V}$ *such that:*

$$Sol(\mathbf{Ax} \leq \mathbf{b}) = \{\mathbf{x} \mid \mathbf{x} = \mathbf{R}\boldsymbol{\lambda}, \boldsymbol{\lambda} \geq 0\ \} + \{\mathbf{x} \mid \mathbf{x} = \mathbf{V}\boldsymbol{\gamma}, \boldsymbol{\gamma} \geq 0, \Sigma\boldsymbol{\gamma} = 1\ \},$$

and $Sol(\mathbf{Ax} \leq \mathbf{0}) = \{\mathbf{x} \mid \mathbf{x} = \mathbf{R}\boldsymbol{\lambda}, \boldsymbol{\lambda} \geq 0\ \}$.

A column of $\mathbf{R}$ is called a *ray*: for any $\mathbf{x}_0 \in Sol(\mathbf{Ax} \leq \mathbf{b})$ and ray $\mathbf{r}$, it turns out that $\mathbf{r}\lambda + \mathbf{x}_0 \in Sol(\mathbf{Ax} \leq \mathbf{b})$ for every $\lambda \geq 0$. A column of $\mathbf{V}$ is called a *vertex*. The set $ConvexHull(\mathbf{V}) = \{\mathbf{x} \mid \mathbf{x} = \mathbf{V}\boldsymbol{\gamma}, \boldsymbol{\gamma} \geq 0, \Sigma\boldsymbol{\gamma} = 1\ \}$, where $\mathbf{V}$ is a matrix or a finite set of vectors, is the convex hull of the vertices, namely the smallest convex set which contains all vertices. An efficient procedure to extract minimal $\mathbf{R}$ and $\mathbf{V}$ is the double description method, also known as the Motzkin-Chernikova-Le Verge algorithm [3,17]. Turning on the LPCheck procedure, the satisfiability test at **Step 2** is performed as part of the construction of the explicit representation of the polyhedron. The maximization problems at **Step 4 (a)** can easily be solved directly on the explicit representation.

Lemma 5. *Consider a characteristic cone* $Sol(\mathbf{Ax} \leq \mathbf{0})$*, and let* $\mathbf{R}$ *be its generating matrix. We have that* $max\{\mathbf{c}^T\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{0}\} = 0$ *iff* $\mathbf{c}^T\mathbf{R} \leq \mathbf{0}$.

Since in our context $\mathbf{c}$ is always zero except for the i^{th} element, which is 1 or -1, we can conclude that a variable $\mathbf{v}_i$ (the i^{th} variable in $\mathbf{v}$) is bounded from above by 0 (resp., bounded from below by 0) iff all values in $row(\mathbf{R}, i)$ are non-positive (resp., non-negative).

Example 6. The Minkowski's form of the homogeneous system in Example 5 is:

$$\begin{pmatrix} 1 & 1 \\ -2 & -1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix}, \lambda_1 \geq 0, \lambda_2 \geq 0$$

Intuitively, the two columns in the generating matrix $\mathbf{R}$ correspond to vectors lying on the two borders of the cone in the graph of Example 5. Using Lemma 5, it is readily checked that when $\mathbf{c}$ is one of $(-1, 0, 0)$, $(0, 1, 0)$, $(0, 0, 1)$ or $(0, 0, -1)$ then $\mathbf{c}^T\mathbf{R} \leq \mathbf{0}$, i.e. x is bounded from below, y from above, and z from both.

2.4 Checking Type Assertions: A Parameterized Approach

In this section, we reason on the parameterized system in (2) by adopting an approach that explicitly considers parameters [14], and which is an extension of the Minkowski's decomposition theorem. However, the complexity of the approach in presence of k parameters is polynomially proportional (by a k^3 factor) to its complexity in absence of parameters. For this reason, we first ask ourselves whether we can build on the results of the last subsection. The LPCheck procedure is sound, and it is also complete except for the ! type. Thus, we are restricted with checking assertions of the form $\mathbf{d} \vdash c \rightarrow x :!$. Under this context, are all typings in $\mathbf{d}$ necessary?

Example 7. Consider $x : \square \vdash z = x, z - y = 2, z + y = 0 \rightarrow x :!, y :!, z :!$. Starting from the involved constraint, by Gaussian elimination, we derive: $x = z$, $y = z-2$, $2z = 2$ and then $z = 1, y = -1, x = 1$. Hence the type assertion is valid.

Notice that we made no use of $x : \square$ in proving validity of the type assertion. This fact can be generalized, in order to get rid of unnecessary parameters.

Theorem 3 (Definiteness). $\mathbf{d} \vdash c \rightarrow x :!$ *is valid iff* $\mathbf{d}|_! \vdash c \rightarrow x :!$ *is valid.*

Let us consider now an example which illustrates the Fourier-Motzkin elimination method for linear inequalities applied in presence of parameters.

Example 8. Consider the constraint c defined as $y + x \leq z, y - x \leq z, z \leq y, 0 \leq z, w \leq z$, and the type declaration $z :!$. We start by isolating variable y in $\phi(z :!) \wedge c$, as shown at **(a)** in the figure below.

$$\begin{array}{rl} & y \leq z - x \\ & y \leq z + x \\ z \leq & y \\ 0 \leq & z \\ & w \leq z \\ & z = a \end{array} \qquad \begin{array}{rl} & \\ z \leq & y \leq min\{z - x, z + x\} \\ & x = 0 \\ 0 \leq & z \\ & w \leq z \\ & z = a \end{array} \qquad \begin{array}{rl} & y = a \\ & x = 0 \\ & w \leq a \\ & z = a \\ 0 \leq & a \end{array}$$

(a) **(b)** **(c)**

Bounds for variable y can then be summarized as: $(*)$ $z \leq y \leq min\{z-x, z+x\}$. Moreover, the bounds $e_1 \leq e_2$ are implied for e_1 expression bounding y from below in $(*)$, and e_2 bounding y from above in $(*)$. Actually, the original set of linear inequalities over y is equivalent to $z \leq y \leq min\{z - x, z + x\}$ plus such bounds. The new inequality set is reported at **(b)** in the figure above. By replacing backward $x = 0$ and $z = a$, we end up with the final system at **(c)** in the figure above, where no further elimination is possible. The final system is feasible when the condition $0 \leq a$ holds. In this system, we have $x = 0, y = a, z = a$. Moreover, $w \leq a$ can be rewritten as: $w = -\lambda_1 + a$ for some $\lambda_1 \geq 0$. Put in a geometrical form, the solution set of $\phi(z :!) \wedge c$ is:

$$\left\{ (x, y, z, w) \mid \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -1 \end{pmatrix} \left(\lambda_1 \right) + \begin{pmatrix} 0 \\ a \\ a \\ a \end{pmatrix} \right\}$$
$$\text{for every } \lambda_1 \geq 0, \quad \text{when } a \geq 0$$

Summarizing, the values of x, y and z are univocally determined once the parameter a has been fixed and the system is feasible. Under the same hypotheses, the value of w is bounded from above (by a), but it is not definite. With our notation, $z :! \vdash c \rightarrow x :!, y :!, z :!, w : \sqcap$ is valid.

The final form reached at the end of the example resembles the Minkowski's form for polyhedra, but with a parameterized vector appearing in the vertex matrix.

Input: a type assertion $\mathbf{d}_1$, a linear constraint c and a sequence of variables $\mathbf{x}$.

Step 0 Define $\mathbf{v} = vars(c)$, $\mathbf{n} = nf(\mathbf{d}_1)$, $\mathbf{d} = \mathbf{n}|_!$.
Step 1 Let $\mathbf{A}_c\mathbf{v} \leq \mathbf{b}$ be the geometric representation of c, and $\mathbf{A_d v} \leq \mathbf{B_d a_d}$ the geometric representation of $\phi(\mathbf{d})$.
Step 1 For the parametric polyhedron $\begin{pmatrix}\mathbf{A}_c\\ \mathbf{A_d}\end{pmatrix}\mathbf{v} \leq \begin{pmatrix}\mathbf{b}_c\\ \mathbf{0}\end{pmatrix} + \begin{pmatrix}\mathbf{0}\\ \mathbf{B_d}\end{pmatrix}\mathbf{a}$, build the generating matrix $\mathbf{R}$ and the sequence $(\mathbf{v^a}(1), \mathbf{C}_1\mathbf{a} \leq \mathbf{c}_1), \ldots, (\mathbf{v^a}(k), \mathbf{C}_k\mathbf{a} \leq \mathbf{c}_k)$
Step 2 For every $x : \tau$ as output from LPCheck
Step 3 If $\tau \neq \Box$ or $x \notin \mathbf{v}$ Then output "$x : \tau$";
Step 4 Else let i such that $x = \mathbf{v}_i$:
(a) Output "x :!" if $row(\mathbf{R}, i) = \mathbf{0}$ and for $1 \leq m < n \leq k$, $\mathbf{v^a}(m)_i = \mathbf{v^a}(n)_i$ over $\mathbf{C}_m\mathbf{a} \leq \mathbf{c}_m$, $\mathbf{C}_n\mathbf{a} \leq \mathbf{c}_n$;
(b) Output "$x : \Box$" otherwise.

Fig. 2. POLYCheck procedure

The generalization of the Minkowski's theorem to parameterized polyhedra is provided in [14] and implemented in the `polylib` library [13].

Theorem 4 (Minkowski's theorem for parameterized polyhedra). *Every parameterized polyhedron can be expressed by a generating matrix* $\mathbf{R}$ *and finitely many pairs* $(\mathbf{v^a}(1), \mathbf{C}_1\mathbf{a} \leq \mathbf{c}_1), \ldots, (\mathbf{v^a}(k), \mathbf{C}_k\mathbf{a} \leq \mathbf{c}_k)$ *where, for* $i = 1..k$, $\mathbf{v^a}(i)$ *is a vector parametric in* $\mathbf{a}$ *and* $Sol(\mathbf{C}_i\mathbf{a} \leq \mathbf{c}_i) \neq \emptyset$*, as follows:*

$$Sol(\mathbf{Ax} \leq \mathbf{b} + \mathbf{Ba}, \mathbf{u}) = \{\mathbf{x} \mid \mathbf{x} = \mathbf{R}\boldsymbol{\lambda}, \boldsymbol{\lambda} \geq 0\ \} + ConvexHull(\{\mathbf{v^u}(i) \mid i = 1..k, \mathbf{C}_i\mathbf{u} \leq \mathbf{c}_i\ \}),$$

and $Sol(\mathbf{Ax} \leq \mathbf{0}) = \{\mathbf{x} \mid \mathbf{x} = \mathbf{R}\boldsymbol{\lambda}, \boldsymbol{\lambda} \geq 0\ \}$.

The vertex matrix is now replaced by a set of pairs where the first element is a parameterized vertex and the second one is its *validity domain*. For a parameter instance $\mathbf{u}$, the vertex matrix is built from the (instantiated) vertices whose validity domain includes $\mathbf{u}$. The special case $k = 0$ models *empty parameterized polyhedra*, which are empty for any instance of the parameters. Now that we have an explicit form for parameterized polyhedra, we need a procedure to test whether a variable is definite for every parameter instance. First, we introduce a notion to test whether two expressions are equal over a polyhedron.

Definition 8. *We say that* $\mathbf{c}_1^T\mathbf{x} + \alpha_1 = \mathbf{c}_2^T\mathbf{x} + \alpha_2$ *over* $\mathbf{Ax} \leq \mathbf{b}$ *if for every* $\mathbf{x}_0 \in Sol(\mathbf{Ax} \leq \mathbf{b})$, $\mathbf{c}_1^T\mathbf{x}_0 + \alpha_1 = \mathbf{c}_2^T\mathbf{x}_0 + \alpha_2$.

Given the Minkowski's form, equality can be checked as follows.

Lemma 6. $\mathbf{c}_1^T\mathbf{x} + \alpha_1 = \mathbf{c}_2^T\mathbf{x} + \alpha_2$ *over* $\mathbf{Ax} \leq \mathbf{b}$ *iff* $(\mathbf{c}_1 \quad \alpha_1) = (\mathbf{c}_2 \quad \alpha_2)$ *or* $Sol(\mathbf{Ax} \leq \mathbf{b}) = \emptyset$ *or, called* $\mathbf{R}$ *and* $\mathbf{V}$ *the generating and vertex matrices of* $\mathbf{Ax} \leq \mathbf{b}$, $(\mathbf{c}_1 - \mathbf{c}_2)^T\mathbf{R} = \mathbf{0}$ *and* $(\mathbf{c}_1 - \mathbf{c}_2)^T\mathbf{V} = (\alpha_2 - \alpha_1)\mathbf{1}^T$.

Notice that checking $(\mathbf{c}_1 \quad \alpha_1) = (\mathbf{c}_2 \quad \alpha_2)$ is not strictly necessary, but if we interpret the three conditions in the lemma from a computational point of view,

the first one provides a very fast test. The next result states that definiteness of a variable x over a parameterized polyhedron amounts to showing that no pair of vertices is such that their projections over x differ for any parameter instance in the non-empty intersection of their domains.

Lemma 7. *Consider the Minkowski's form of a non-empty parameterized polyhedron as in Theorem 4. Every* $\mathcal{S}_{\mathbf{u}} = \{\mathbf{c}^T\mathbf{x} \mid \mathbf{x} \in Sol(\mathbf{Ax} \leq \mathbf{b}+\mathbf{Ba}, \mathbf{u})\}$ *is empty or a singleton iff* $\mathbf{c}^T\mathbf{R} = \mathbf{0}$ *and for* $1 \leq m < n \leq k$, $\mathbf{c}^T\mathbf{v^a}(m) = \mathbf{c}^T\mathbf{v^a}(n)$ *over* $\mathbf{C}_m\mathbf{a} \leq \mathbf{c}_m, \mathbf{C}_n\mathbf{a} \leq \mathbf{c}_n$.

Example 9. Consider a parameterized polyhedron over parameters $(a \;\; b)$ and variables $(x \;\; y)$ with generating matrix $\mathbf{0}$, and with pairs of vertices and domains:

$$(\begin{pmatrix} a \\ b \end{pmatrix}, b \geq a \geq 0) \qquad (\begin{pmatrix} a \\ a \end{pmatrix}, a \geq b \geq 0) \qquad (\begin{pmatrix} a \\ a+b \end{pmatrix}, a \geq 0 \wedge b \geq 0).$$

Let us reason about definiteness of variables x and y by using Lemma 7. x is definite, since $a = a$ over any polyhedron. Consider now y. For the first two vertices, we have $b \neq a$, or, in vectorial notation, $(0 \;\; 1)\begin{pmatrix} a \\ b \end{pmatrix} + \alpha_1 \neq (1 \;\; 0)\begin{pmatrix} a \\ b \end{pmatrix} + \alpha_2$ where $\alpha_1 = \alpha_2 = 0$. Since the intersection of the two domains, namely $a = b \geq 0$, is not empty, by Lemma 6 we proceed by computing its generating and vertex matrices. The vertex matrix is $\mathbf{0}$, so we simply have:

$$Sol(a = b \geq 0) = \{(a, b) \mid \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}(\lambda_1), \lambda_1 \geq 0\},$$

and we have to test: $(\begin{pmatrix} 0\;1 \end{pmatrix} - \begin{pmatrix} 1\;0 \end{pmatrix})\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \end{pmatrix}$, where (0) is $(\alpha_2 - \alpha_1)\mathbf{1}^T$. Therefore, a and b are equal over $a = b \geq 0$. Consider now the first and the third vertex. Since $b \neq a + b$, we compute, as before, the Minkowski's form of the intersection of their domains:

$$Sol(b \geq a \geq 0) = \{(a, b) \mid \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix}, \lambda_1 \geq 0, \lambda_2 \geq 0\}.$$

Then we test: $(\begin{pmatrix} 0\;1 \end{pmatrix} - \begin{pmatrix} 1\;1 \end{pmatrix})\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} -1\;0 \end{pmatrix}$, which differs from the expected $\begin{pmatrix} 0\;0 \end{pmatrix} = (\alpha_2 - \alpha_1)\mathbf{1}^T$ for $\alpha_2 = \alpha_1 = 0$. Summarizing, by Lemma 6, b and $a + b$ are not equal over $b \geq a \geq 0$, and then, by Lemma 7, y is not definite.

Lemmas 6 and 7 provide us with a checking procedure for definiteness. The overall procedure, called POLYCheck, is shown in Fig. 2. POLYCheck terminates and is sound and complete for inferring validity of type assertions.

Theorem 5 (POLYCheck - soundness and completeness). *Let* $\mathbf{d}_1$ *be a type declaration and* c *a linear constraint. If the sequence* $\mathbf{d}_2$ *is provided as output by* POLYCheck*, the type assertion* $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ *is valid. Conversely, assume that* $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ *is valid. Then there exists a sequence* $\mathbf{d}$ *provided as output by* POLYCheck *such that* $\mathbf{d} \geq_t nf(\mathbf{d}_2)$.

2.5 Extensions to Strict Inequalities and to Disequalities

So far, we considered equality and non-strict inequality primitive constraints. A generalized linear constraint admits primitive constraint over the operators $<$, $>$ (strict inequalities) and $\neq$ (disequalities). Without any loss of generality, we write a generalized constraint as $c \wedge \bigwedge_{i=1}^{m} e_i \neq \alpha_i$, where c is a linear constraint and for $i = 1..m$, $e_i \neq \alpha_i$ is a disequality. We now extend type assertions to admit generalized constraints. The next result shows that validity of type assertions for a satisfiable generalized constraint can be reduced to validity of the type assertions over the linear constraint obtained by removing the disequalities in it.

Theorem 6. *Let $g = c \wedge \bigwedge_{i=1}^{m} e_i \neq \alpha_i$ be a satisfiable generalized linear constraint. $\mathbf{d}_1 \vdash g \rightarrow \mathbf{d}_2$ is valid iff $\mathbf{d}_1 \vdash c \rightarrow \mathbf{d}_2$ is valid.*

Checking satisfiability of g is easily accomplished when computing the explicit form of polyhedra. By independence of negative constraints [12], it reduces to show that $Sol(\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c) \neq \emptyset$ and that every hyperplane $e_i = \alpha_i$ does not include the polyhedron $Sol(\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c)$, i.e., by Definition 8 that $e_i = \alpha_i$ over $\mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c$ is false. Lemma 6 provides us with a procedure to check it.

3 Moding CLP Programs

Modes for pure logic programs assign to every predicate argument an input-output behavior. Input means that the predicate argument is ground on calls. Output means that it is ground on answers. As discussed in the introduction, groundness (i.e., definiteness) is restrictive in the CLP context. Based on types, we can extend the notion of moding to upper and/or lower bounds as well.

Definition 9 (moding). *A* mode *for a n-ary predicate p is a function d_p from $\{1, \ldots, n\}$ to $\mathcal{BT} \times \mathcal{BT}$. We write d_p as $p(\tau_1 \times \mu_1, \ldots, \tau_n \times \mu_n)$, where $d_p(i) = (\tau_i, \mu_i)$ for $i = 1..n$.*

A mode *for a CLP($\mathcal{R}$) program P is a set of modes, one for each predicate in P. For an atom $p(\mathbf{x})$, we write $p(\mathbf{x} : \boldsymbol{\tau} \times \boldsymbol{\mu})$ to denote that $\mathbf{x}$ is the collection of variables occurring in the atom, and $p(\boldsymbol{\tau} \times \boldsymbol{\mu})$ is the mode of p.*

By fixing a predicate argument mode to !×! or to ⋆×! we get back to the logic programming input-output behavior, respectively denoted by $+$ and $-$. Several notions of moding have been proposed [1]. We consider here well-moding by extending it to CLP($\mathcal{R}$) programs.

Definition 10 (well-moding). *Let P be a CLP($\mathcal{R}$) program. A clause $p_0(\mathbf{x}_0 : \boldsymbol{\mu}_0 \times \boldsymbol{\tau}_{n+1}) \leftarrow c, p_1(\mathbf{x}_1 : \boldsymbol{\tau}_1 \times \boldsymbol{\mu}_1), \ldots, p_n(\mathbf{x}_n : \boldsymbol{\tau}_n \times \boldsymbol{\mu}_n)$ in P is well-moded if for $i = 1..n+1$, the type assertion $\mathbf{x}_0 : \boldsymbol{\mu}_0, \ldots, \mathbf{x}_{i-1} : \boldsymbol{\mu}_{i-1} \vdash c \rightarrow \mathbf{x}_i : \boldsymbol{\tau}_i$ is valid. P is well-moded if every clause in it is well-moded.*

Example 10. The `MORTGAGE` program is well-moded with moding `mortgage(`!×!`,` ⊓×!`,` !×!`,` ⋆×!`)`, which models the first two queries in the introduction. For

clause `(m1)` we have to show: `P` :!, `T` : $\sqcap$, `R` :! $\vdash$ `T = 0, B = P` $\rightarrow$ `P` :!, `T` :!, `R` :!, `B` :! which is immediate. For clause `(m2)`, called c the constraint `T >= 1, NP = P + P * 0.05 - R, NT = T - 1`, we have to show: `P` :!, `T` : $\sqcap$, `R` :! $\vdash c \rightarrow$ `NP` :!, `NT` : $\sqcap$, `R` :! and `P` :!, `T` : $\sqcap$, `R` :!, `NP` :!, `NT` :!, `B` :! $\vdash c \rightarrow$ `P` :!, `T` :!, `R` :!, `B` :! which are both readily checked. Analogously, `MORTGAGE` is well-moded with the moding `mortgage(`$\star \times \square$`,` $\sqcap \times !$`,` $\square \times \square$`,` $\square \times \square$`)`, which models the third query in the introduction.

We recall that the operational semantics of CLP consists of a transition system from states to states. A state is a pair $\langle Q \| c \rangle$ where Q is a query and c is a constraint, called the constraint store. Initial states are of the form $\langle Q \| \texttt{true} \rangle$. Final states (if any) are of the form $\langle \varepsilon \| c \rangle$, where ε is the empty query. Well-moding extends to states $\langle Q \| c' \rangle$ by considering the program clause $p \leftarrow c', Q$, where p is a fresh 0-ary predicate.

Definition 11. *A state* $\langle \leftarrow c, p_1(\mathbf{x}_1 : \boldsymbol{\tau}_1 \times \boldsymbol{\mu}_1), \ldots, p_n(\mathbf{x}_n : \boldsymbol{\tau}_n \times \boldsymbol{\mu}_n) \| \ c' \rangle$*, with* $n \geq 0$*, is well-moded if for* $i = 1..n$ *the type assertion* $\mathbf{x}_1 : \boldsymbol{\mu}_1, \ldots, \mathbf{x}_{i-1} : \boldsymbol{\mu}_{i-1} \vdash (c \wedge c') \rightarrow \mathbf{x}_i : \boldsymbol{\tau}_i$ *is valid. A query* Q *is well-moded if the state* $\langle Q \| \ \texttt{true} \rangle$ *is well-moded.*

Widely studied properties of well-moding in logic programming include persistency along derivations, call pattern characterization and computed answer characterization. They are at the basis of several program analysis, transformation and optimization techniques. The next result shows that the mentioned properties hold for the proposed extension of well-moding to CLP($\mathcal{R}$). By a left-derivation we mean a derivation via the leftmost selection rule.

Theorem 7. *Let* P *be a well-moded CLP($\mathcal{R}$) program and* $Q = \ \leftarrow c, p_1(\mathbf{x}_1 : \boldsymbol{\tau}_1 \times \boldsymbol{\mu}_1), \ldots, p_n(\mathbf{x}_n : \boldsymbol{\tau}_n \times \boldsymbol{\mu}_n)$ *a well-moded query.*

[persistency] *Every state selected in a left-derivation of* P *and* Q *is well-moded.*
[call patterns] *For every state of the form* $\langle \leftarrow q(\mathbf{x} : \boldsymbol{\tau} \times \boldsymbol{\mu}), R \| \ c' \rangle$ *selected in a left-derivation of* P *and* Q*,* $\vdash c' \rightarrow \mathbf{x} : \boldsymbol{\tau}$ *is valid.*
[answers] *For every final state* $\langle \leftarrow \varepsilon \| \ c' \rangle$*,* $\vdash c' \rightarrow \mathbf{x}_1 : \boldsymbol{\mu}_1, \ldots, \mathbf{x}_n : \boldsymbol{\mu}_n$ *is valid.*

Based on these properties, we provide next two examples of the kind of analyses that well-moding allows for.

Example 11. The two queries from the introduction $\leftarrow$ `mortgage(100, 5, 20, B)` and $\leftarrow$ `3 <= T, T <= 5, mortgage(100, T, 20, B)` are well-moded with the moding `mortgage(!`$\times$`!,` $\sqcap \times !$`,` $! \times !$`,` $\star \times !$`)`. By Theorem 7, we conclude definiteness of balance in the answer constraint store. The third query from the introduction $\leftarrow$ `0 <= B, B <= 10, 15 <= R, R <= 20, mortgage(P, 5, R, B)` is well-moded with the moding `mortgage(`$\star \times \square$`,` $\sqcap \times !$`,` $\square \times \square$`,` $\square \times \square$`)`. By Theorem 7, we conclude boundedness of principal in the answer constraint store.

Example 12. The full version of the `MORTGAGE` program takes the interest rate as a further predicate argument.

```
(n1)  mortgage(P,T,I,R,B) ←          (n2)  mortgage(P,T,I,R,B) ←
        T = 0,                               T >= 1,
        B = P.                               NP = P + P * I - R,
                                             NT = T - 1,
                                             mortgage(NP,NT,I,R,B).
```

However, this leads to a non-linear constraint appearing in clause `(n2)`. How can we reason on it? We exploit the call pattern characterization property of well-moding by factoring out the `P * I` term.

```
(n2′)  mortgage(P,T,I,R,B) ←          (mu)  mult(P,I,M) ←
         T >= 1,                              P * I = M.
         NP = P + M - R,
         NT = T - 1,
         mult(P, I, M),
         mortgage(NP,NT,I,R,B).
```

Consider now as if the predicate `mult` is a built-in of the system, and the input-output properties of Theorem 7 are guaranteed for the mode `mult( !×!, !×!, ⋆×!)`. The rest of the program, namely clauses `(n1)` and `(n2′)`, is readily checked to be well-moded with moding `mortgage(!×!, ⊓×!, !×!, !×!, ⋆×!)`. Therefore, for every call to `mult` the first and the second arguments are definite, and then the non-linear constraint `P * I = M` becomes linear at run-time.

4 Related Work and Conclusions

A class of formulas, called *parametric queries*, is investigated in [9]. It includes formulas $\exists a \forall \mathbf{v}\ c \to x \simeq a$, where $\simeq \in \{\leq, =, \geq\}$, or, with our notation, type assertions of the form $\vdash c \to x : \tau$. The approach switches from the problem of checking $max\{\mathbf{c}^T\mathbf{v} \mid \mathbf{A}_c\mathbf{v} \leq \mathbf{b}_c\} \leq a$ to its dual form $max\{0 \mid \mathbf{y}^T\mathbf{A}_c = \mathbf{c}, a = \mathbf{y}^T\mathbf{b}_c + q, \mathbf{y} \geq \mathbf{0}, q \geq 0\} = 0$, namely on checking feasibility of $\mathbf{y}^T\mathbf{A}_c = \mathbf{c}, a = \mathbf{y}^T\mathbf{b}_c + q, \mathbf{y} \geq \mathbf{0}, q \geq 0$. However, as soon as general type assertions $\mathbf{d}_1 \vdash c \to \mathbf{d}_2$ are considered, switching to the dual form yields a non-linear problem.

The maximization of a linear function over a parameterized polyhedra is the subject of *(multi)parametric linear programming*. The solution of the problem can be expressed as a piecewise linear function [7] of the parameters. Therefore, an approach alternative to the extraction of the Minkowski's form is to compute (for each variable to be typed) the *max* and *min* functions of a parameteric linear problem and then to compare them on each pair of breaks they are defined on.

Definiteness analysis for CLP($\mathcal{R}$) has been investigated in [2,4,5,8] using abstract interpretation. Compared to well-moding, those approaches *infer* boolean expressions relating definiteness of predicate arguments. E.g., an inferred $x \to y$ for $p(x, y)$ states that if x is definite when $p(x, y)$ is called then y is definite when it is resolved. However, the mentioned approaches restrict to consider equality constraints only, hence cannot be complete as the type assertion framework.

Summarizing the contribution of the paper, we have introduced a type system for (generalized) linear constraints over the reals that is able to reason about

upper bounds, lower bounds and definiteness properties of variables. The problem of checking validity of type assertions has been investigated and solved by proposing two specialized decision procedures. Type assertions are the basic tool for extending well-moding from logic programming to CLP($\mathcal{R}$). We implemented the checking procedure for well-moding, including LPCHECK and POLYCHECK, in standard C++, relying on the `polylib` library [13] for the calculation of the Minkowski's form of (parameterized) polyhedra (sources and extended technical report at `http://www.di.unipi.it/~ruggieri/software`). Although a comprehensive assessment over larger programs has to be pursued, our preliminary tests provide us with confidence on the efficiency of the approach in practice.

References

1. Apt, K.R.: From Logic Programming to Prolog. Prentice-Hall, Englewood Cliffs (1997)
2. Baker, N., Søndegaard, H.: Definiteness analysis for CLP($\mathcal{R}$). In: Gupta, G., et al. (eds.) Australian Computer Science Conference, pp. 321–332 (1993)
3. Chernikova, N.V.: An algorithm for finding the general formula for non-negative solutions of systems of linear inequalities. U.S.S.R. Computational Mathematics and Mathematical Physics 5, 228–233 (1965)
4. Codish, M., Genaim, S., Søndegaard, H., Stuckey, P.: Higher-precision groundness analysis. In: Codognet, P. (ed.) ICLP 2001. LNCS, vol. 2237, pp. 135–149. Springer, Heidelberg (2001)
5. de la Banda, M.G., Hermenegildo, M., Bruynooghe, M., Dumortier, V., Janssens, G., Simoens, W.: Global analysis of constraint logic programs. ACM Transactions on Programming Languages and Systems 18(5), 564–614 (1996)
6. Dolzmann, A., Sturm, T., Weispfenning, V.: Real quantifier elimination in practice. In: Matzat, B.H., Greuel, G.-M., Hiss, G. (eds.) Algorithmic Algebra and Number Theory, pp. 221–248. Springer, Berlin (1998)
7. Gal, T.: Postoptimal Analyses, Parametric Programming, and Related Topics, 2nd edn., de Gruyter, Berlin, Germany (1995)
8. Howe, J.M., King, A.: Abstracting numeric constraints with boolean functions. Information Processing Letters 75(1-2), 17–23 (2000)
9. Huynh, T., Joskowicz, L., Lassez, C., Lassez, J.-L.: Practical tools for reasoning about linear constraints. Fundamenta Informaticae 15(3-4), 357–380 (1991)
10. Jaffar, J., Maher, M.J.: Constraint logic programming: A survey. Journal of Logic Programming 19, 20, 503–581 (1994)
11. Jaffar, J., Michaylov, S., Stuckey, P., Yap, R.: The CLP($\mathcal{R}$) language and system. ACM Transactions on Programming Languages and Systems 14(3), 339–395 (1992)
12. Lassez, J.-L., McAllon, K.: A canonical form for generalized linear constraints. Journal of Symbolic Computation 13(1), 1–24 (1992)
13. Loechner, V.: Polylib: a library for manipulating parameterized polyhedra, Version 5.22.3 (2007), `http://icps.u-strasbg.fr/polylib/`
14. Loechner, V., Wilde, D.K.: Parameterized polyhedra and their vertices. International Journal of Parallel Programming 25, 525–549 (1997)
15. Murty, K.G.: Linear Programming. John Wiley & Sons, Chichester (1983)
16. Schrijver, A.: Theory of Linear and Integer Programming. J. Wiley & Sons, Chichester (1986)
17. Le Verge, H.: A note on Chernikova's algorithm. Technical Report 635, IRISA, Campus Universitaire de Beaulieu, Rennes, France (1992)